```
SELECT DISTINCT fragmentID, doctype, lastModifiedTime, title, creator,
pagetype FROM meta.metaall  where  KEYWORD LIKE 'SERVER%' and PAGETYPE LIKE
'FRAGMENT' and iscommit = 1
```

An example DASL output to above query:

```
<?xml version="1.0"?>
<d:multistatus xmlns:d  ="DAV:" xmlns:f="Franklin:">
  <f:responsesummary>
     <f:start>1</f:start>
     <f:end>1</f:end>
     <f:total>1</f:total>
  </f:responsesummary>/
  <d:response>
      <d:href>http://franklinserver/46b3e60dccbcd84db007777-tfrg.xml
      </d:href>
     <d:propstat>
       <d:prop>
        <f:FRAGMENTID>46b3e60dccbcd84db007777-tfrg.xml</f:FRAGMENTID>
        <f:DOCTYPE>TEXTFRAGMENT</f:DOCTYPE>
        <f:LASTMODIFIEDTIME:                                    /f:LASTMODIFIEDTIME>
        <f:TITLE>Netfinity Highlights</f:TITLE>
        <f:CREATOR>Joe Doe</f:CREATOR>
        <f:PAGETYPE>FRAGMENT</f:PAGETYPE>
       </d:prop>
      </d:propstat>
   </d:response>
</d:multistatus>
```

If the number of results is larger than the result set requested by the Search UI, the meta-data store writes the full results into a cache file and only encodes the requested number into the DASL output. The cache file is named using an expression that encodes the query and the *sessionId* of the user. When the Search UI requests the "Next" set of results for the same query, the meta-data store does not re-execute the query. Instead it consults the cache file and extracts the appropriate next set of results. This caching scheme saves the meta-data store from executing the same query several times if the user if simply navigating within the same result set.

Note that if the contents where to change in DB2, the user does not see the updated results until he re-executes the original query without the "Next" or "Previous" flags.

Lock Management

When the dispatcher receives a `LOCK` command from the Editor UI, it creates a lock for a fragment or servable and sends the lock to the meta-data store to save in DB2. The lock information, namely `LOCKTOKEN`, `LOCKEDOWNER`, and `LOCKTIME`, is stored in the `META.LOCK` table of the following format:

Column Name	Data Type	Default	Key	Index
FRAMENTID	CHAR(56)		PK	
LOCKOWNER	VARCHAR(50)			
LOCKTIME	TIMESTAMP			
LOCKTOKEN	VARCHAR(34)			

When the dispatcher receives an UNLOCK command from the Editor UI, it issues the unlock command to the meta-data store. The meta-data store deletes the record from the META.LOCK table.

The Content Store – Daedalus (a.k.a Trigger Monitor)

This section describes how the Franklin project has extended three of the Daedalus handlers to enable the system to manage XML fragments and XSL style sheets. For the full Daedalus API documentation, read http://w3.watson.ibm.com/~challngr/papers/daedalus/index.html.

Daedalus is written in pure Java and implements *handlers* as pre-defined actions performed on the various configurable resources. Flexibility is achieved via Java's dynamic loading abilities, by more sophisticated configuration of the resources used by Daedalus, and through the use of *handler* preprocessing of input data. Most entities defined in a configuration file implement a public Java interface. Users may create their own classes to accomplish localized goals, and specify those classes in the configuration file. This permits run-time flexibility without requiring sophisticated efforts on the part of most users, since default classes are supplied to handle the most common situations.

For Franklin, we have created our own classes to implement three handlers: the Extension Parser, the Dependency Parser, and the Page Assembler.

Extension Parser

Within Franklin, Daedalus manages different types of files differently based on their extensions. Servables, simple, compound, and index fragments, style sheets and multimedia assets are all treated slightly differently in the publishing flow.

The Franklin Extension Parser takes in a name of a fragment, and returns an extension used in the Daedalus configuration files to specify actions to take during the publish process:

```
123445-trfg.xml     => tfrg  (text fragment)
123445-bfrg.xml     => bfrg  (binary wrapper fragment)
123445-ifrg.xml     => ifrg  (index fragment)
123445-tsrv.xml     => tsrv  (text servable fragment)
123445-sfrg.xml     => sfrg  (style sheet wrapper fragment)
web_index.xsl       => xsl   (style sheet)
```

The appropriate behavior for each type of fragment (e.g. source-to-sink, assemble-to-sink) is defined in the Daedalus configuration files. Generally, only servables are assembled to the sink.

Dependency Parser

The Franklin Dependency Parser reads through an XML objects that has been checked in and detects two types of dependencies:

1. Servables and fragments can include subfragments, these get stored as an edge of type "composition" in the Daedalus Object Dependency Graph (ODG).
2. Compound fragments include an associated content file, this dependency gets stored as an edge type "composition" in the ODG.
3. Servables can include style sheets, these get stored as an edge type "stylesheet" in the ODG.

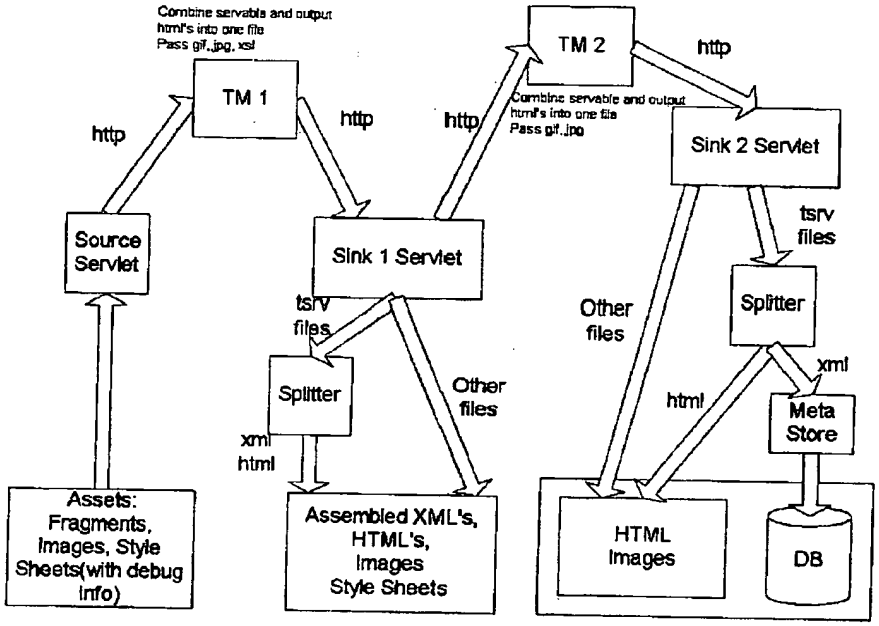
Dependencies are considered to point from the subfragments to the fragments that include them. For binary wrappers, one composition dependency points from the wrapper to the fragment that includes it, and another points from the wrapper to the binary data file that it wraps. For stylesheets, a composition dependency points from the wrapper to the stylesheet, and a stylesheet dependency points from the stylesheet to the servable that uses it.

Page Assembler

The Franklin Page Assembler expands a servable by including the contents of all included subfragments, and combines the resulting XML with the one or more style sheets using LotusXSL to produce HTML output files. The extension of each of the resulting files is determined from the stylesheet names (e.g. web_xxx_**html**.xsl). The assembled XML and all the resulting HTML files are written to one file, which is later split up in the Dispatcher, and the HTML files are written to the appropriate directories in the sink or server.

Chaining of Trigger Monitors

Currently, two Trigger Monitors are used in the publish process. They share an ODG, and the sink of the first one is the source of the second, creating a publishing chain. The following diagram shows the set-up of the Content store in its entirety:



When a fragment is checked in to the Content store, it is added to the shared ODG, and a publish command is issued to the first TM. The TM reads the fragment XML from the source servlet, uses the extension parser to find its extension, and then uses the dependency parser to find dependencies to add to the ODG. The page assembler then pulls in the contents of the fragment's subfragments, and if the fragment is a servable, combines it with its stylesheets to produce the output HTMLs. The servable XMLs, output HTMLs, binary files, and stylesheets are sent to the servlet specified as the sink of the first TM.

When a servable has been approved, a publish command on the servable fragment is issued to the second TM. It is reassembled and recombined with its XSLs, and the resulting XML and HTMLs are published to the second sink servlet. Binary files (such as images) are also published to the second sink. This is where the web server pulls the final HTML and image files from.

Example application

- managing Netfinity pages at ibm.com

Summary

Appendix 1: Error Codes

```
# Status code (101) indicating the server is switching protocols
# according to Upgrade header. (SC_SWITCHING_PROTOCOLS)

X1 = 101

# Status code (200) indicating the request succeeded normally. (SC_OK)

G200 = 200
P200 = 200
OK = 200

# Status code (201) indicating the request succeeded and created
# a new resource on the server. (SC_CREATED)

X3 = 201

# Status code (202) indicating that a request was accepted for
# processing, but was not completed. (SC_ACCEPTED)

X4 = 202

# Status code (203) indicating that the meta information presented
```

```
# by the client did not originate from the server.
(SC_NON_AUTHORITATIVE_INFORMATION)

X5 = 203

# Status code (204) indicating that the request succeeded but that
# there was no new information to return. (SC_NO_CONTENT)

X6 = 204

# Status code (205) indicating that the agent <em>SHOULD</em> reset
# the document view which caused the request to be sent. (SC_RESET_CONTENT)

X7 = 205

# Status code (206) indicating that the server has fulfilled
# the partial GET request for the resource. (SC_PARTIAL_CONTENT)

X8 = 206

# Status code (300) indicating that the requested resource
# corresponds to any one of a set of representations, each with
# its own specific location. (SC_MULTIPLE_CHOICES)

X9 = 300

# Status code (301) indicating that the resource has permanently
# moved to a new location, and that future references should use a
# new URI with their requests. (SC_MOVED_PERMANENTLY)

X10 = 301

# Status code (302) indicating that the resource has temporarily
# moved to another location, but that future references should
# still use the original URI to access the resource. (SC_MOVED_TEMPORARILY)

X11 = 302

# Status code (303) indicating that the response to the request
# can be found under a different URI. (SC_SEE_OTHER)

X12 = 303

# Status code (304) indicating that a conditional GET operation
# found that the resource was available and not modified. (SC_NOT_MODIFIED)
```

X13 = 304

Status code (305) indicating that the requested resource
MUST be accessed through the proxy given by the
<code>Location</code> field. (SC_USE_PROXY)

X14 = 305

Status code (400) indicating the request sent by the client was
syntactically incorrect. (SC_BAD_REQUEST)

THIS IS THE GENERAL (DEFAULT) ERROR RETURNED WHEN ANYTHING BREAKS

#check c102 to make sure it belongs in this area (400)
C101 = 400
C102 = 400
C103 = 400
C123 = 400
C124 = 400
D104 = 400
D110 = 400
P101 = 400
V101 = 400
F100 = 400
F101 = 400
F102 = 400
F103 = 400
F104 = 400
F105 = 400
R101 = 400
R112 = 400
R102 = 400
R103 = 400
R105 = 400
D101 = 400
D111 = 400
D145 = 400
G103 = 400

Status code (401) indicating that the request requires HTTP
authentication. (SC_UNAUTHORIZED)

G101 = 401
U101 = 401
U102 = 401
U103 = 401

L101 = 401
L102 = 401
G104 = 401
# Status code (402) reserved for future use. (SC_PAYMENT_REQUIRED)

X17 = 402

Status code (403) indicating the server understood the request
but refused to fulfill it. (SC_FORBIDDEN)

G102 = 403
D123 = 403
# Status code (404) indicating that the requested resource is not
# available. (SC_NOT_FOUND)

X19 = 404

Status code (405) indicating that the method specified in the
<code>Request-Line</code> is not allowed for the resource
identified by the <code>Request-URI</code>.
(SC_METHOD_NOT_ALLOWED)

X20 = 405

Status code (406) indicating that the resource identified by the
request is only capable of generating response entities which have
content characteristics not acceptable according to the accept
headerssent in the request. (SC_NOT_ACCEPTABLE)

F108 = 406

Status code (407) indicating that the client MUST first
authenticate itself with the proxy. (SC_PROXY_AUTHENTICATION_REQUIRED)

X22 = 407

Status code (408) indicating that the client did not produce a
requestwithin the time that the server was prepared to wait. (SC_REQUEST_TIMEOUT)

X23 = 408

Status code (409) indicating that the request could not be
completed due to a conflict with the current state of the
resource. (SC_CONFLICT)

X24 = 409

Status code (410) indicating that the resource is no longer
available at the server and no forwarding address is known.
This condition SHOULD be considered permanent. (SC_GONE)

X25 = 410

Status code (411) indicating that the request cannot be handled
without a defined <code>Content-Length</code>. (SC_LENGTH_REQUIRED)

X26 = 411

Status code (412) indicating that the precondition given in one
or more of the request-header fields evaluated to false when it
was tested on the server. (SC_PRECONDITION_FAILED)

X27 = 412

Status code (413) indicating that the server is refusing to process
the request because the request entity is larger than the server is
willing or able to process. (SC_REQUEST_ENTITY_TOO_LARGE)

X28 = 413

Status code (414) indicating that the server is refusing to service
the request because the <code>Request-URI</code> is longer
than the server is willing to interpret. (SC_REQUEST_URI_TOO_LONG)

X29 = 414

Status code (415) indicating that the server is refusing to service
the request because the entity of the request is in a format not
supported by the requested resource for the requested method.
(SC_UNSUPPORTED_MEDIA_TYPE)

X30 = 415

Status code (500) indicating an error inside the HTTP server
which prevented it from fulfilling the request. (SC_INTERNAL_SERVER_ERROR)

X31 = 500

Status code (501) indicating the HTTP server does not support
the functionality needed to fulfill the request. (SC_NOT_IMPLEMENTED)

X32 = 501

```
# Status code (502) indicating that the HTTP server received an
# invalid response from a server it consulted when acting as a
# proxy or gateway. (SC_BAD_GATEWAY)

X33 = 502

# Status code (503) indicating that the HTTP server is
# temporarily overloaded, and unable to handle the request. (SC_SERVICE_UNAVAILABLE)

X34 = 503

# Status code (504) indicating that the server did not receive
# a timely response from the upstream server while acting as
# a gateway or proxy. (SC_GATEWAY_TIMEOUT)

X35 = 504

# Status code (505) indicating that the server does not support
# or refuses to support the HTTP protocol version that was used
# in the request message. (SC_HTTP_VERSION_NOT_SUPPORTED)

X36 = 505

# Error code in server.dispatcher.Dispatcher

# D104 = Error in Dispatcher.doPost()
# D110 = Fragment Type not Specified or incorrect
# P101 = Error in Dispatcher.putParseRequest()
# V101 = Error validating user

# Error codes in server.Fragment

# F100 = Error in Fragment.fragment2XML()
# F101 = Error opening Fragment.XML2fragment()
# F102 = Error parsing XML file in Fragment.XML2fragment()
# F103 = Error calling readNode("+element+")
# F104 = Error calling getElementValue
# F105 = Error calling getElementType
# F120 = Cannot close StringWriter

# Error codes in server.TextUtils

# R101 = Error in TextFile.read("+filename+")
# R112 = Error in TextFile.readTextFileWOException("+filename+")
# R102 = Error in TextUtils.createDOMfromFile("+xmlfile+")
```

```
# R103 = TextUtils.createDomFromFile SAX exception
# R105 = TextUtils.createDomFromFile IO exception
# R112 = Error in TextFile.readTextFileWOException("+filename+")


# Error codes in server.dispatcher.DomUtils

# D101 = DomUtils.documentToTuniverse TXDOM Exception
# D111 = Error in DomUtils.documentToUniversal
# D123 = Missing or invalid sessionID on checkin

# Error codes in server.dispatcher.Users

# these have destination ERROR_USER

# U101=  User + username + not defined
# U102 = Wrong password  for user + username
# U103 = User with sessionid + sessionId + not defined

# this has destination ERROR_LOG

# U110 = Users.methodname IO exception

# Error codes in server.dispatcher.checkIn

# C103 = Error in document2String
# C102 = Checkin Error
# C101 = Users.checkProvledge error

# DE111 = Delete Error

# G200 = successful get
# P200 = successful put

# Locking errors
# L101 = Lock tokens do not match
# L102 = missing lock token


#MISC
# F108 = invalid FragmentID
# C123 = error in fragment2XML
# C124 = Failed saving content to metadata store

# G104 = Authorization String Empty
# D145 = error parsing input stream
```